LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-234181

# Asynchronous Event-Driven Particle Algorithms

A. Donev

August 31, 2007

SIMULATION: Transactions of the Society for Modeling and Simulation International

## Disclaimer

# Asynchronous Event-Driven Particle Algorithms

Aleksandar Donev[1]

[1]*Lawrence Livermore National Laboratory,*
*P.O.Box 808, Livermore, CA 94551-9900*

## Abstract

We present, in a unifying way, the main components of three asynchronous event-driven algorithms for simulating physical systems of interacting particles. The first example, hard-particle molecular dynamics (MD), is well-known. We also present a recently-developed diffusion kinetic Monte Carlo (DKMC) algorithm, as well as a novel stochastic molecular-dynamics algorithm that builds on the Direct Simulation Monte Carlo (DSMC). We explain how to effectively combine event-driven and classical time-driven handling, and discuss some promises and challenges for event-driven simulation of realistic physical systems.

## I. INTRODUCTION

There is a wide range of particle systems from computational science problems that are best simulated using *asynchronous event-driven* (AED) algorithms. Examples include: *molecular dynamics* (MD) for systems of hard particles [4, 18] such as disordered packings [3], granular materials [7], colloids [8, 15] and particle-laden flows [17]; *kinetic Monte Carlo* (KMC) [12] simulation of diffusion-limited reactions (DKMC) [13] such as nucleation, growth, and coarsening during epitaxial growth [14], diffusion quantum Monte Carlo [16], nuclear reactions [20], bio-chemical reactions [19], and self-assembly of nano-structures [9]; *direct simulation Monte Carlo* (DSMC) [1] for micro hydrodynamics [**?** ], granular flows [7], and plasma flows [10]; *contact dynamics* for modeling systems of rigid bodies computer graphics [11]; and many others. As of yet unexplored are *multi-scale* and *multi-physics* algorithms such as combined flow and diffusion with (bio)chemical reactions.

In this work we will focus on a class of particle-based problems that are very common in computational materials science and are well-suited for AED simulation. Specifically, we will focus on the simulation of large systems of mobile particles interacting with *short-range pairwise* (two-body) potentials (forces). Our goal will be to reveal the common building blocks of these simulations (e.g., event queues, neighbor searches), but also to highlight the components that are problem specific (e.g., event prediction and processing). We will present these components in some detail for three specific examples: hard-particle MD, DKMC, and DSMC. Through the discussion of these examples we will demonstrate the undeniable advantages of AED algorithms, but we will also reveal the difficulties with using AED algorithms for realistic models.

### A. Background

We consider the simulation of the time evolution of a collection of $N$ interacting particles in $d$-dimensions, starting from some initial condition. At any point in time, the system $\mathcal{Q} = (\mathbf{Q}, \mathcal{B})$ is characterized by the *configuration* $\mathbf{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_N)$, containing at least the centroid positions $\mathbf{r}_i$ for every particle $i$, and the additional global information $\mathcal{B}$, which may involve variables such as boundary conditions or external fields. The number of particles $N$ may also vary with time. For each particle $i$ we may consider an arbitrary number of

attributes in addition to the position of the centroid, for example, $\mathbf{q}$ may also contain the linear and/or angular velocity, the orientation and/or the chemical specie (shape, charge, mass, internal composition) of particle $i$. Typically each particle configuration $\mathbf{q}_i$ will at least contain an integer that identifies its *specie* $1 \leq s_i \leq N_s$, and some information will be shared among all particles belonging to the same specie (ex., the charge or mass). In particular, the symmetric *interaction table* $\mathcal{I}_{\alpha\beta}$ stores $N_s(N_s+1)/2$ logical (true or false) entries that specify whether species $\alpha$ and $\beta$ interact or not.

Two particles $i$ and $j$ are *overlapping* only if a certain (generalized) distance between them $d_{ij}(\mathbf{q}_i, \mathbf{q}_j) \geq 0$ is less than some *cutoff distance* or *diameter* $D_{ij}$. Overlapping particles react with each other in an application specific manner. Typically the type of reaction and $D_{ij}$ depends (only) on the species of the two particles and $D_{ij}$, but there may also be dependencies on time or some other external field parameters. For example, for (additive) hard spheres $D_{ij} = (D_i + D_j)/2$ and the type of reaction is (hard-core) repulsion. For a non-interacting pair of particles one may set $D_{ij} = 0$. Particles may also overlap with boundaries of the simulation domain, such as hard walls or reactive surfaces, however, typically the majority of interactions are among particles.

We will assume that the time evolution (motion) of the system is mostly smooth with the exception of certain discrete *events*, which lead to discontinuous changes in the configuration of a certain collection of particles. Events may involve a single particle $i$, such as the change of the internal state of the particle (e.g., decay reactions, spin flips, sudden changes in the particle velocity). Events may also involve pairs of particles, for example, the *collision* (exchange of momentum) between two particles $i$ and $j$, or a chemical reaction between two overlapping particles $i$ and $j$ leading to the creation and/or destruction of particles. For now we will ignore events involving more than two particles. Some events may also involve global variables in $\mathcal{B}$ and thus implicitly affect all of the particles.

We will refer to simulations as *event-driven* (also called *discrete event simulations* [6]) if the state of the system is evolved discontinuously in time from one event to another, *predicting* the time of the next event whenever an event is processed. This is in contrast with the more common *time-driven* simulations, where the state changes continuously in time and is evolved over a sequence of small time steps $\Delta t$, discovering and processing events at the end of the time step. Typically, particle-based time-driven algorithms are not exact, in the sense that they may miss events when the time step used is too large. Therefore,

the time step must be smaller than the estimated slowest time scales in the problem. This leads to large inefficiencies when there are multiple dynamically-changing time scales. On the other hand, event-driven algorithms automatically adjust the time step.

We will focus on *asynchronous* event-driven (AED) algorithms. In asynchronous algorithms, there is a global simulation time $t$, typically the time when the last processed event occurred, and each particle is at a different point in time $t_i \leq t$, typically the last time it participated in an event. This is to be contrasted to synchronous event-driven algorithms, where all of the particles are at the same time $t$. One of the most important examples of a synchronous event-driven algorithm in materials science is the $n$-fold algorithm for performing kinetic (dynamic) Monte Carlo simulations [12]. The applicability and efficiency of synchronous algorithms hinge on the fact that the state of the system does not change in-between events, as is common in lattice models where the positions of the particles are discrete. For example, atoms may stay in the immediate vicinity of the crystal lattice sites and only sometimes hop to nearby sites. In the types of problems that we will consider, the positions of the particles will be continuous and continuously changing even in-between events. Therefore, we will not discuss synchronous algorithms further. It is important to point out that even in cases where the evolution of the system consists entirely of discrete jumps (e.g., Markov chain transitions), asynchronous algorithms may be more efficient than synchronous ones [9]. In general, one may combine the two algorithms by using the more general asynchronous algorithm at the top level but treating a subset of the particles synchronously, as if they are a super-particle with complex internal structure.

### B. Model Examples

Atomistic or molecular-level modeling is one of the foundations of computational materials science. The two most popular types of algorithms used in the simulation of materials are molecular dynamics (MD) and Monte Carlo (MC) algorithms. Monte Carlo algorithms are often used to study static equilibrium properties of systems, however, here we focus on dynamic or kinetic Monte Carlo, where the time evolution of a system is modeled just like in MD. For our purposes, the only important difference between the two is that MD is a deterministic algorithm, in which deterministic equations of motions are solved, and Monte Carlo is a stochastic procedure in which sample paths from an ensemble of weighted

paths is generated. In both cases one typically averages over multiple trajectories, starting with different initial conditions and/or using a different random number seed. From the perspective of AED algorithms, this means that random number generators (RNGs) are involved in the determination of the time certain events occur as well as in the actual processing of those events.

The very first *molecular dynamics* (MD) calculations simulated a system of hard disks and hard spheres and used an AED algorithm. The hard-sphere system is a collection of non-overlapping spheres (or disks), contained within a bounded region, and each moving with a certain velocity $\mathbf{v}_i = \dot{\mathbf{r}}_i$. Pairs of spheres collide and the colliding particles bounce off elastically, preserving both linear momentum and energy. Many generalizations can be considered, for example, the spheres may be growing in size and/or the particles may be nonspherical [3], the collisions may not be perfectly elastic [7], some of the particles may be tethered to each other to form structures such as polymer chains [18], etc. The general features are that particles move along simple deterministic paths (such as straight lines) in-between binary collisions, which are the primary type of event. The events take zero time and involve deterministic changes of the velocities of the colliding particles. The ballistic motion of the particles is described by Newton's equations of motion, $m\dot{\mathbf{v}}_i = \mathbf{F}_i$ (i.e., deterministic ODEs). Event-driven MD algorithms for hard particles are discussed in considerable detail in Ref. [4] and elsewhere, here we only present the essential components.

*Direct simulation MC* (DSMC) [1] is an MC algorithm that tries to mimic MD for fluids. We will consider DSMC as a fast alternative to MD, even though it can also be viewed as a particle-based MC method for solving the Boltzmann equation in dilute fluids. From our perspective, DSMC is an approximate variant of MD in which the particle collisions are not processed exactly, rather, particle collisions are stochastic and (attempt to) follow the same probability distributions as would have exact MD. Specifically, nearby particles are randomly chosen to undergo stochastic collisions and exchange momentum and energy, thus leading to local conservation laws and hydrodynamic behavior. DSMC is applicable in cases when the structure of the fluid and the detailed motions of all of the particles do not matter, as is the case with solvent molecules (e.g., water) in fluid flow problems or large-scale granular flows [7]. Traditionally DSMC has been implemented using a time-driven approach, in which at each time step particles are first propagated in a ballistic (convective) fashion, and then a certain number of stochastic particle collisions among nearby particles are processed. Here

we describe a novel AED algorithm for DSMC, and demonstrate how it can be integrated with AED MD in order to replace the expensive MD with cheaper DSMC for some of the species (e.g., solvent molecules).

The motion of the particles in-between events is not always deterministic. In particular, an important class of problems concerns diffusing particles, that is, particles whose velocity changes randomly very frequently (i..e, they make many small steps in random directions). The motion of the particles is probabilistic, in the sense that the probability $c(\mathbf{r}, t + \Delta t)$ of finding a particle at a given position $\mathbf{r}$ at a certain time $\Delta t$, assuming it started at the (space and time) origin, is the solution to the time-dependent diffusion equation $\partial_t c = D\boldsymbol{\nabla}^2 c$ (a deterministic PDE), where $D$ is the particle diffusion coefficient. A variety of *reactions* may occur when a pair of particles collides, for example, particles may repel each other (colloids [15]), they may stick or begin merging together (paint suspensions), or they may undergo a chemical reaction that consumes the reacting particles and produces zero, one, two, or possibly more new particles (a wide range of diffusion-limited reactions in materials [13, 19] and biological systems [19]). Several *approximate* event-driven KMC algorithms have been used in the past for this problem [19]. Here we will describe a recently-developed *exact* AED algorithm for simulating a collection of diffusing hard particles [13]. It is worth pointing out that one may also consider particles whose trajectories are a combination of ballistic and diffusive motion, that is, motion that is described by Langevin's equations (or other stochastic ODEs or even PDEs). In that sense, we will see that both the MD and MC algorithms share many common features.

## II. A GENERAL AED PARTICLE ALGORITHM

In this work we focus on systems where particles only interact with nearby particles. We will formalize this by defining a geometric hierarchy of regions around a given particle. These particle proximity hierarchies are at the core of *geometry-specific* (GS) aspects of AED simulation, which can be reused for different *application-specific* (AS) rules for moving and interacting the particles. We will assign a *hard core* $\mathcal{C}_i$ to each particle such that a particle may overlap with another particle only if their cores overlap. For (additive) hard spheres, the core is nothing more than the particle itself. Next, we *protect* particle $i$ against other particles by enclosing it inside a *protective region* $\mathcal{P}_i$, $\mathcal{C}_i \subseteq \mathcal{P}_i$, that is typically disjoint from

the majority of other protective regions. Finally, we assume that every protective region $i$ is contained within a *neighborhood region* $\mathcal{N}_i$, $\mathcal{P}_i \subseteq \mathcal{N}_i$. The set of *neighbors* of $i$ consists of the particles $j$ whose neighborhood regions intersect $\mathcal{N}_i$, $\mathcal{N}_i \cap \mathcal{N}_i \neq 0$, and which are of a specie interacting with the specie of particle $i$, $\mathcal{I}_{s_i s_j} = \mathcal{T}$.

We will assume that when a particle does not interact with other particles we can easily follow its time evolution (motion), that is, given the current configuration $\mathbf{q}_i(t)$, we can probabilistically determine the position at a later time $\mathbf{q}_i(t + \Delta t)$. This is a *single-particle problem* and can typically be solved analytically. For example, in MD the particle trajectory is a unique (i.e., deterministic) straight path, $\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \mathbf{v}_i \Delta t$, while in diffusion problems it is the solution to a (stochastic) Langevin equation of motion. Event-driven algorithms are efficient because they use such analytic solutions to quickly propagate particles over potentially large time steps as long as they are far enough from other (interacting) particles. We will also assume that one can solve two-body problems for the case when two particles are isolated from other particles but may interact with each other. These two-body problems are typically much more difficult to solve (quasi) analytically. Specific examples will be given later.

### A. The Event Loop

An AED algorithm consists of processing a sequence of time-ordered events. Each particle $i$ must store some basic information needed to predict and process events associated with it. The particle time $t_i$ specifies the last time the configuration of particle $i$ was updated, $t_i \leq t$, where $t$ is the current simulation time. Some particles may be time-driven and thus not have their own event prediction. The rest of the particles are event-driven and each such particle stores a prediction for its *impending event* $(t_e, p, \nu)$, specified via the predicted time of occurrence $t_e$ (a floating point number), the event *partner* $p$ (an integer), and the event *qualifier* (type of event) $\nu$ (also an integer). The partner $p$ could be some pre-specified invalid value to identify time-driven particles. Note that the *event schedules* must be kept symmetric at all times, that is, if particle $i$ has an impending event with $j$, then particle $j$ must have an impending event with $i$. A particle may store multiple event predictions, in order to avoid re-predicting events if the impending event is invalidated, however, we will not explicitly handle this possibility due to the complications it introduces.

The exact interpretation of $p$ and $\nu$, for a given particle $i$, is application- and geometry-specific. Some common types of events can be pre-specified by reserving certain values of the event partner $p$:

$p = 0$ An *update* of the event prediction for $i$, not requiring an update of $\mathbf{q}_i$. The value $\nu = 0$ means that $\mathbf{q}_i$ has not changed since the last prediction for $i$ (thus allowing stored information from previous predictions to be reused if needed), $\nu = 1$ means that an event occurred which did not alter the geometry (for example, the position of $i$ is the same but its velocity changed), while $\nu = -1$ means that this particle was just inserted into the system and a geometry update is necessary as well.

$p = i$ A single-particle event that requires an update of $\mathbf{q}_i$. The special value $\nu = 0$ denotes a simple time advance of $i$ without any additional event processing, $\nu < 0$ denotes an event that does not change the geometry (for example, only the velocity of a particle changes), and $\nu > 0$ is used for additional AS events that may also change the geometry (e.g., particle decay).

$1 \le p \le N$ **and** $p \ne i$ An unprocessed *binary reaction* between particles $i$ and $j = p$, with additional AS information about the type of reaction stored in $\nu$, for example, elastic collision, a certain chemical reaction, etc.

$p = \infty$ A "boundary" event requiring the update of the particle geometry. If $\nu = 0$ then only the protective region $\mathcal{P}_i$ needs to be updated, if $\nu = -1$ then the neighborhood $\mathcal{N}_i$ needs to be updated (collision with a *virtual boundary*), $\nu < -1$ denotes collisions with pre-specified boundaries (such as hard walls), and $\nu > 0$ specify AS boundary events (such as collisions with reactive surfaces).

$p = -\infty$ Denotes an invalid event, meaning that this particle is not in the event queue and is handled separately, for example, it is time-driven.

It is important to point out that we are not suggesting that an actual implementation needs to use integers to identify different types of events. In an object-oriented framework events may be represented as objects that inherit from a base event class and have methods to handle them, with the base implementation providing handlers for certain pre-defined (single, pair, and boundary) types of events. We do not discuss here the possible ways to organize

an inheritance hierarchy of classes for AED simulations, since such a hierarchy involves multiple complex components, notably a module for handling boundary conditions in static and dynamic environments, a module for handling static and dynamic particle geometry (overlap, neighborhoods, neighbor searches, etc.), an event-dispatcher, a visualization module, application-specific modules for event scheduling and handling, etc.

Algorithm 1 represents the main *event loop* in the AED algorithm, which processes events one after the other in the order they occur and advances the global time $t$ accordingly. It uses a collection of other auxiliary geometry-specific (GS) or application-specific (AS) steps, as marked in the algorithm outline. Specific examples of various GS and AS steps are given in the next section.

#### *1. Non-Particle Events*

In a variety of applications the majority of events can be associated with a specific particle, and we schedule one event per particle in the event queue. However, sometimes there may be events that are associated with a (possible large) group of particles, or events that are not specifically associated with a particle. We consider these non-particle events as application-specific "external" events in Algorithm 1.

An important example of such an event are *time step events*. Namely, some group of particles may not be propagated asynchronously using the event queue, instead, the particles in the group may be updated synchronously, for example, in regular time intervals $\Delta t$. There may in fact be multiple such groups each with their own timestep, for example, each specie might have its own time step. These should also be ordered in time and the next one chosen as the external event in Step 2 in Algorithm 1. A separate priority queue may be used for ordering the external events. In general, there may be an event queue of events associated with particles, with cells, with species, etc. These may be separate queues that are joined at the top or they can be merged into a single event-queue.

### B. Near-Neighbor Search

All large-scale particle-based algorithms use various geometric techniques to make the number of neighbors of a given particle $O(1)$ instead of $O(N)$. Reference [4] provides exten-

**Algorithm 1** Process the next event in the event queue.

1. Find (query) the top of the event queue (usually a heap) to find the next particle $i$ to have an event with $p$ at $t_e$. Note that steps marked as (AS+C) below may reorder the queue and/or cycle back to this step.

2. Find the next "external" event to happen at time $t_{ex}$, possibly using an additional event queue (AS).

3. If $t_{ex} < t_e$ then process the external event (AS) and cycle back to step 1.

4. Remove $i$ from the event queue and advance the global simulation time $t \leftarrow t_e$.

5. If $p = 0$ and $\nu = -1$ then build a new $\mathcal{N}_i$ and then check if $i$ overlaps with any of its new neighbors (GS). If it does, process the associated reactions (AS+C), otherwise build a new $\mathcal{P}_i$ as in step 8a.

6. Else if $p = i$ then update the configuration of particle $i$ to time $t$ using a single-particle propagator (AS), and set $t_i \leftarrow t$. If $\nu \neq 0$ then process the single-particle event (AS+C). If $\nu > 0$ then search for overlaps as in step 5.

7. Else if $1 \leq p \leq N$ then update the configuration of particles $i$ and $j = p$ using a two-particle propagator (AS), set $t_i \leftarrow t$ and $t_j \leftarrow t$, and then process the binary reaction between $i$ and $j$ (AS+C). This may involve inserting particle $j$ back into the queue with $t_e = t$, $p = 0$, $\nu = 0$.

8. Else if $p = \infty$, then update $\mathbf{q}_i$ and $t_i$ as in step 6. If $\nu > 0$ then process the boundary event (AS+C), otherwise

   (a) If $\nu = 0$ then update $\mathcal{P}_i$ (AS+GS), typically involving an iteration over the neighbors of $i$.

   (b) Else if $\nu = -1$ then update $\mathcal{N}_i$ and identify the new neighbors of particle $i$ (GS).

   (c) Else if $\nu < -1$ process the geometry-induced boundary event (GS+AS).

9. Predict a new $t_e$, $p$, and $\nu$ for particle $i$ by finding the minimal time among the possible events listed below. Each successive search needs to only extend up to the current minimum event time, and may return an incomplete prediction $t_e > t$, $p = 0$, $\nu = 0$, where $t_e$ provides a lower bound on the actual event time.

   (a) When particle $i$ leaves $\mathcal{P}_i$ or $\mathcal{N}_i$ (AS).

sive details (and illustrations) of these techniques for hard spheres and ellipsoids; here we summarize only the essential components.

### *1. Linked List Cell (LLC) Method*

The most basic technique is the so-called *linked list cell* (LLC) method. The simulation domain, typically an orthogonal box, is partitioned into $N_c$ cells, typically cubes. Each particle $i$ stores the cell $c_i$ to which its centroid belongs, and each cell $c$ stores a list $\mathcal{L}_c$ of all the particles it contains (usually we also store the total number of particles in the cell). Given a particle and a range of interaction, the lists of potential neighbors is determined by scanning through the neighboring cells. For maximal efficiency the cell should be larger than the largest range of interaction so that only the nearest-neighbor cells need to be searched. There are more sophisticated neighbor search methods developed in the computational geometry community, such as using (colored) quad/oct-trees, however, we are not aware of their use in AED implementations, likely because of the implementation complexity. This is an important subject for future research.

It is important to point out that in certain applications the cells themselves play a crucial role in the algorithm, typically as a means to provide mesoscopic averages of physical variables (averaged over the particles in a given cell) used to switch from a particle-based model to a continuum description. For example, in PIC (particle-in-cell) algorithms for plasma simulation, the cells are used to solve for background electric fields using FFT transforms [10]. In DSMC, the algorithm stochastically collides pairs of particles that are in the same cell. In some applications, events may be associated with the cells themselves, instead of or in addition to the events associated with particles [5]. Usually the same cells are used for both neighbor searches and averaging for simplicity, however, this may not be the optimal choice in terms of efficiency.

For a method that only uses the LLC method for neighbor searches, the neighborhood region $\mathcal{N}_i$ is composed of the (typically $3^d$, where $d$ is the dimensionality) cells that neighbor $c_i$. The protection region $\mathcal{P}_i$ may be a simple geometric region like a sphere inscribed in $\mathcal{N}_i$ (sphere of diameter smaller than the cell size), it may be that $\mathcal{P}_i \equiv c_i + \mathcal{C}_i$, or $\mathcal{P}_i \equiv \mathcal{N}_i$, or it may be that $\mathcal{P}_i \equiv \mathcal{N}_i \cap (c_i + \mathcal{C}_i)$.

*2. Near-Neighbor List (NNL) Method*

Another neighbor search method is the *near-neighbor list* (NNL) method, which is described for hard particles in Ref. [4]. The idea is to use as $\mathcal{N}_i$ a region that (when it is created) is just an enlargement of the particle by a certain scaling factor $\mu > 1$. When $\mathcal{N}_i$ is created the method also creates a (linked) list of all the neighborhoods that intersect it, to form NNL($i$) (hard walls or other boundaries may also be near neighbors). This list of (potential) *interactions* can then be reused until the particle core $\mathcal{C}_i$ protrudes outside of $\mathcal{N}_i$. This reuse leads to great savings in situations where particles are fairly localized. Note that the LLC method is still used in order to create $\mathcal{N}_i$ and NNL($i$) even if NNLs are used, in order to keep the maximal cost of pairwise searches at $O(N)$ instead of $O(N^2)$. In some situations (such as mixed MD/DSMC simulations as we describe later) one may use NNLs only for a subset of the particles and use the more traditional LLCs for others.

*3. Cell Bitmasks*

In our implementation, in addition to the list of particles $\mathcal{L}_c$, each cell $c$ stores a *bitmask* $\mathcal{M}_c$ consisting of $N_{\text{bits}} > N_s + 4$ bits (bitfields). These bits may be one (set) or zero (not set) to indicate certain properties of the cell, specifically, what species of particles the cell contains, whether the cell is event or time driven, and to specify boundary conditions. Bit $\gamma$ in the bitmask $\mathcal{M}_c$ is set if the cell $c$ may contain a particle of specie $\gamma$. The bit is set whenever a particle of specie $\gamma$ is added to the cell. When performing a neighbor search for a particle $i$, cells not containing particles of species that interact with specie $s_i$ are easily found (by OR'ing the cell masks with the $s_i$'th row of $\mathcal{I}_{s_i s_j}$) and are simply skipped. This speeds can significantly speed up the neighbour searches in cases where not all particles interact with other particles.

For the purposes of a combined event-driven time-driven algorithm one may also need to distinguish those cells where particles are treated using a fully event-driven (ED) scheme. We use one of the bits in the bitmasks, bit $\gamma_{ED}$, to mark *event-driven (ED) cells* , and the choice of such cells is in general application specific. For example, for diffusion problems, cells with a high density may be treated more efficiently using time-driven (small hopping) techniques while areas of low density may be treated more efficiently using the asynchronous

event-driven algorithm. In our combined MD with DSMC algorithm cells near non-DSMC particles are event-driven while those containing only DSMC particles are treated as time-driven cells. Note that all of the cell bitmasks should be reset and then re-built (i.e., refreshed) periodically. In our experience, it is necessary to introduce at least one "sticky" bit $\gamma_{st}$ that is not cleared but rather persists (has memory), and is initialized to zero (not set) at the beginning of the simulation.

The cell masks can also be used to specify partitionings of the simulation domain. This is very useful in specifying more general boundary conditions in situations when the event-driven simulation is embedded inside a larger domain that is not simulated explictly. For example, a molecular dynamics simulation may be embedded in a multiscale solver where the surrounding space is treated using a continuum method (finite element or finite volume, for example) coupled to the particle region through an intermediary boundary layer. Similar considerations apply in parallelization via domain decomposition, where the simulation domain simulated by a single processing element (PE) or logical process (LP) is embedded inside a larger region where other domains are simulated by other PEs/LPs.

We classify the cells as being *interior, boundary, and external cells* (a specific illustrative example is given in Fig. 1). Our implementation uses bits in the cell bitmasks to mark a cell as being boundary (bit $\gamma_B$), or external (bit $\gamma_P$), the rest are interior. Interior cells are those for which complete boundary conditions are specified and that cannot be directly affected by events occuring outside of the simulation domain (the interior cells are divided into event-driven and time-driven as discussed earlier). Boundary cells surround the interior cells with a layer of cells of thickness $w_B \geq 1$ cells, and they represent cells that are affected by *external events* (i.e., events not simulated directly). External cells are non-interior cells that are not explicitly simulated, rather, they provide a boundary condition/padding around the interior and boundary cells. This layer must be at least $w_B$ cells thick, and the cells within a layer of $w_B$ cells around the simulation domain (interior together with boundary cells) are marked as both external and boundary cells (e.g., these could be ghost cells in parallel simulation). All of the remaining cells are purely external cells and simply ignored by the simulation.

## III. MODEL EXAMPLES

In this section we present the handling of the various AS and GS steps in Algorithm 1 for three specific model applications.

### A. Event-Driven Molecular Dynamics (EDMD)

Hard-particle molecular dynamics is one of the first applications of AED algorithms in computational science, and is discussed in more detail in Ref. [4] and references therein. Hard-sphere MD has been used extensively in simulations of physical systems over the past decades and is also one of the few particle AED algorithm that has been studied in the discrete-event simulation community, as the *billiards* problem. Because of this rich history and extensive literature we only briefly discuss EDMD focusing on how it fits our general framework.

The basic type of event in EDMD are *binary collisions*, which alter particle momenta, typically using elastic collision laws (conservation of momentum and energy). Collisions are assumed to have no duration and (very unlikely) triple collisions are broken up into a sequence of binary ones. In-between collisions particles move ballistically along simple trajectories such as straight lines (force-free motion) or parabolas (constant-acceleration motion). For hard spheres, event time predictions are based on (algebraic) methods for finding the first root of a polynomial equation (linear, quadratic or quartic [8]). For particle shapes that include orientational degrees of freedom, such as ellipsoids, numerical root finding techniques need to be used [4].

When LLCs are used, the main type of boundary event are *cell transfers*, which occur when the centroid of a particle $i$ collides with the boundary of the cell $c_i$. If the cell is at the boundary, a *unit cell change* occurs for periodic boundaries (i.e., wrapping around the torus), and *hard-wall collisions* occur for hard-wall boundaries. When NNLs are used, cell transfers do not have to be processed (i.e., one can set $\mathcal{P}_i \equiv \mathcal{N}_i$), unless it is important to have accurate LLCs [as in DSMC, where $\mathcal{P}_i \equiv \mathcal{N}_i \cap (c_i + \mathcal{C}_i)$].

The main AS steps in Algorithm 1 for (classical) MD simulations are:

- Step 6 consists of updating the particle position, and possibly also velocity, ballistically.

- Step 7 consists of updating the positions of each of the particles separately, as in step 6, and then updating their velocities taking into account the collisional exchange of momentum.

- For collisions with hard walls in Step 8, the particle velocity is updated accordingly. Cell transfers in step 8c consist of updating the LLCs by removing the particle from its current cell and inserting it into its new cell (found based on the direction of motion of the centroid). If the new cell is across a periodic boundary, the centroid is translated by the appropriate lattice cell vector (if NNLs are used, this may require updating information relating to periodic boundaries for each interaction).

- Step 9 is the most involved and time consuming:

  - In step 9a, the time of the next cell transfer is predicted based on the centroid velocity. If NNLs are used, then the time of (virtual) collision of the core $\mathcal{C}_i$ with the interior wall of $\mathcal{N}_i$ is also calculated.

  - In step 9c, predictions are made for binary collisions between particle $i$ and each of the particles in neighboring cells, or between $i$ and each of the particles in NNL($i$).

### B. Stochastic Event-Driven Molecular Dynamics (SEDMD)

We have recently developed a novel AED algorithm for simulating hydrodynamics at the molecular level that combines DSMC, which is a method for simulating hydrodynamic transport, with event-driven molecular dynamics (EDMD). The algorithm replaces some of the deterministic collisions in EDMD with stochastic collisions and we term it Stochastic EDMD (SEDMD). The algorithm is described in more detail in Ref. [2].

First we describe how to transform the traditional time-driven DSMC algorithm [1] into an event-driven algorithm, and then combine this algorithm with EDMD. Finally, we explain how to achieve higher efficiency by reverting the DSMC component to time-driven handling.

### 1. DSMC for Hydrodynamics

The DSMC algorithm can also be viewed as an approximation to molecular dynamics in cases when the internal structure of the fluid, including the true equation of state, is not important. In particular, this is the case when simulating a solvent in applications such as the simulation of large polymer chains in solution. The exact trajectories of the solvent particles do not really matter, and what really matters are the (long time and long range) hydrodynamic interactions that arise because of local energy and momentum exchange (viscosity) and conservation (Navier-Stokes equations). Any method that simulates the correct momentum transfer localized at a sufficiently small scale is a good replacement for full-scale MD, and can lead to great computational savings when a large number of solvent molecules needs to be simulated.

DSMC [1] achieves local momentum exchange and conservation by performing a certain number of stochastic collisions between randomly chosen pairs of particles that are inside the same cell. The collision rate inside a cell containing $N_L$ particles is proportional to $N_L(N_L-1)$ with a pre-factor that can be based on theory or fitted to mimic that of the full MD simulation. For hard spheres, the probability of choosing a particular pair $ij$ is proportional to the relative velocity $v_{ij}^{\mathrm{rel}}$, and typically a rejection technique (null-method technique) is used when choosing pairs. Specifically, the collision rate is made proportional to the maximal possible relative velocity $v_{\mathrm{rel}}^{\mathrm{max}}$, and a randomly chosen pair $ij$ is rejected or accepted with probability $v_{ij}^{\mathrm{rel}}/v_{\mathrm{rel}}^{\mathrm{max}}$. A *pair rejection* involves a small calculation and a random number generation and is thus rather inexpensive, as long as the acceptance probability is not too small, which can typically easily be achieved by a judicious (but still rigorous) choice for $v_{\mathrm{rel}}^{\mathrm{max}}$.

Traditionally DSMC is performed using a time-driven approach: The particles are first propagated ballistically by a certain time step $\Delta t$ and then sorted into cells accordingly, and then an appropriate number of stochastic collisions are carried out. Typically only a fraction (10-25%) of the particles actually undergo a collision, and the rest of the particles are propagated needlessly. Furthermore, the existence of a finite step leads to errors of order $\Delta t^2$, in addition to the errors inherent in DSMC that are of order $\Delta x^2$, where $\Delta x$ is the size of the cells. Therefore, $\Delta t$ must be small enough so that particles move only a fraction of the cell size during one step.

*2. AED Implementation of DSMC*

An alternative event-driven implementation of DSMC explicitly predicts and process cell transfers, just as in the MD algorithm. Particles positions are thus only updated when needed, and there is no time step error. The DSMC particles are represented as a non-interacting specie $\delta$, $\mathcal{I}_{\delta\delta} = \mathcal{F}$, so that the MD algorithm does not predict binary collisions for the DSMC particles. Instead, stochastic binary collisions are added as an external Poisson event of the appropriate rate. One approach is to maintain a global time-of-next-DSMC-collision $t_{sc}$ to determine when a stochastic collision is attempted, and to use *cell rejection* to select a host cell for the collision. The rate of DSMC collisions is chosen according to the cell with maximal occupancy $N_L^{\max}$, and a randomly chosen cell of occupancy $N_L$ is accepted with probability $N_L(N_L-1)/\left[N_L^{\max}(N_L^{\max}-1)\right]$.

Since the DSMC fluid is perfectly compressible, the maximal cell occupancy can be quite high for very large systems, and this leads to decreasing acceptance probability as the size of the system increases. One can avoid cell rejections by using an asynchronous approach. Each cell can store its own time-of-next-DSMC-collision, which is updated whenever the cell occupancy changes. A separate event-queue is used to order these *cell event* times. The top of this *cell event queue* is used as the time of the next "external" event. Other possibilities exist, as commonly used in traditional KMC simulations [12]. For example, the cells could be grouped in lists based on their occupancy and then an occupancy chosen first (with the appropriate weight), followed by selection of a cell with that particular occupancy.

Most of the AS steps in Algorithm 1 for DSMC are shared with MD. The different steps are associated with the processing of stochastic collisions:

- In Step 2 $t_{ex}$ is the time of the the next DSMC collision. If a rejection-free technique is used the cell at the top of the cell event queue is chosen, otherwise, a cell is selected randomly and accepted or rejected based on its occupancy. If the cell is rejected, $t_{ex} \leftarrow \infty$.

- In Step 3 a pair of particles $i$ and $j$ is selected from the previously-chosen cell, and accepted or rejected for collision based on the relative velocity. If accepted, momentum and energy are stochastically exchanged among the particles and they are moved to the top of the event queue with $t_e = t$, $p = 0$, and $\nu = 1$.

We have validated that the event-driven algorithm produces the same results as the time-driven one by comparing against published DSMC results for plane Poiseuille flow of a rare gas.

### *3. Stochastic Event-Driven Molecular Dynamics (SEDMD)*

The event-driven DSMC algorithm has few advantages over a time-driven approach, which are outweighed by the (implementation and run-time) cost of the increased algorithmic complexity. However, the AED variant of DSMC is very similar to AED hard-sphere MD, and therefore it is relatively simple to combine DSMC with MD in an event-driven framework. This enables the simulation of systems such as colloids or hard-sphere bead-chain polymers [18] in solution, where the solute particles are treated using MD, and the less-important solvent particles are treated approximately using DSMC. The solvent-solute interaction is still treated with MD. Similar studies have already been carried out using time-driven MD for the solute particles and a simplified variant of DSMC that approximates the solute particles as point particles and employs multi-particle stochastic collisions [**?** ].

We have implemented such a combined algorithm, which we term Stochastic Event-Driven Molecular Dynamics (SEDMD) [2]. The implementation is almost identical to classical hard-sphere MD, with the addition of a new *DSMC specie* $\delta$ for which $\mathcal{I}_{\delta\delta} = \mathcal{F}$. That is, the DSMC "hard spheres" freely interpentrate each other, but collide as usual with other species. The algorithm is much more efficient than pure MD, however, it is still not as efficient as classical time-driven DSMC. The two main causes of this are the overhead of the event queue operations in the AED variant of DSMC (note that the queue needs to be updated after every stochastic collision), and also the cost of neighbor searches. Namely, for each DSMC particle the nearby cells need to be searched in Step 9c to make sure they do not contain any solute particles (recall that cell bitmasks are used to efficiently implement this). In cases when most of the cells contain only DSMC particles, this can introduce significant overheads.

This inefficiency is best corrected by combining time-driven with event-driven handling. Specifically, only those those cells that neighbor cells that contain non-DSMC particles are marked as event-driven (ED), the rest are time-driven (TD). DSMC particles outside the marked region are treated more efficiently, using time-stepping and without any neighbor

searches, while the DSMC and MD particles inside the marked region are treated as in MD (with the addition of stochastic collisions among DSMC particles). In cases when the solute particles are much larger than the solvent particles, NNLs are used with a special technique called bounded sphere-complexes [4] to handle neighbor searches for the large particles. Whenever DSMC particles transfer from a TD to an ED cell they are inserted into the event queue with an immediate update event, and if needed their neighborhood region and NNL is constructed. Similarly, when DSMC particles transfers from an ED to a TD cell they are removed from the event queue and their neighborhood region and NNL is destroyed.

The time-driven particles are updated together with a fixed time step $\Delta t$. These *time step events* are treated as a special external event and thus processed in correct time order with the events scheduled for the particles in the combined MD/DSMC region. Stochastic collisions are only processed at time step events, exactly as in traditional DSMC algorithms.

#### *4. Open Boundary Conditions*

In simulations of polymer chains in solution in three dimensions, a very large number of solvent particles is required to fill the simulation domain. The majority of these particles are far from the polymer chain and they are unlikely to significantly impact or be impacted by the motion of the polymer chain. These particles do not need to be simulated explicitly, rather, we can think of the polymer chain and the surrounding DSMC fluid as being embedded into an infinite reservoir of DSMC particles which enter and leave the simulation domain following the appropriate distributions.

Such *open (Grand Canonical) boundary conditions* (BC) are often used in multi-scale (coupled) simulations. The combination of a partially time-driven algorithm and an unstructured (ideal gas) DSMC fluid makes it very easy to implement open BCs by inserting DSMC particles in the cells surrounding the simulation domain only at time-step events, based on very simple distributions. At the beginning of a time step event, after possibly rebuilding the cell masks, the time-driven DSMC particles are propagated as usual. If there are external cells, (trial) *reservoir particles* are then inserted into the cells that are both external and boundary. The trial particles are thought to be at a time $t - \Delta t$, and are propagated by a time step $\Delta t$ to the current simulation time. Only those particles that move into a non-external cell are accepted and converted into real particles. Following the

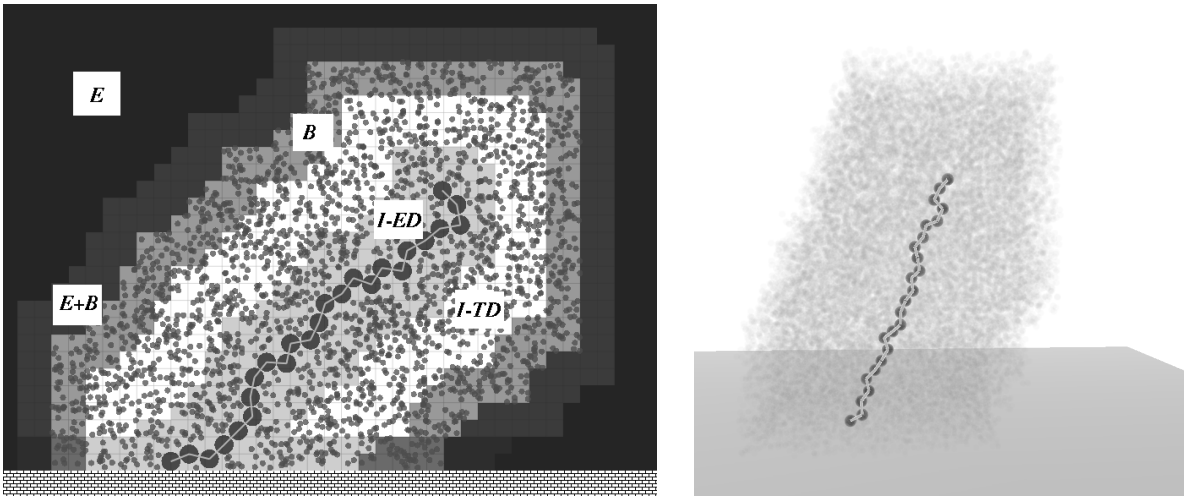


Figure 1: The partitioning of the domain into interior (I) [either event-driven (ED) or time-driven (TD)], boundary (B), and external (E) cells in two (left) and three (right) dimensions for a polymer chain of 25 beads tethered to a hard wall. The cells are shaded in different shades of gray and labeled in the two-dimensional illustration. The DSMC particles are also shown.

insertion of reservoir particles stochastic collisions are processed in each cell as usual.

Figure 1 provides an illustration of the division (masking) of the cells for the simulation of a tethered polymer in two and three dimensions. The division of the cells into event-driven, interior, boundary and external cells is rebuilt periodically during the simulation. This rebuilding may only happen at the beginning of time steps, and requires a synchronization of all of the particles to the current simulation time, a complete rebuilding of the cell bitmasks, and finally, a re-initialization of the event processing. Importantly, particles that are in purely external cells are removed from the simulation and those that are in event-driven cells are re-inserted into the event queue scheduled for an immediate update event.

### C. Diffusion Kinetic Monte Carlo (DKMC)

An exact AED algorithm for kinetic MC simulation of a collection of diffusing particles was recently proposed in Ref. [13]. The main difference with MD is that the equation of motion of the particles is not a deterministic but rather a stochastic ODE. The algorithm simulates a trajectory that is sampled from the correct probability distribution. For pure diffusion with transport coefficient $D$, the probability $c(\Delta\mathbf{r}, \Delta t)$ of finding an isolated point

particle at a displacement $\Delta\mathbf{r}$ at time $\Delta t$ is the Green's function for the (time-dependent) diffusion equation, $\partial_t c = D\boldsymbol{\nabla}^2 c$, with no additional boundary conditions. In three dimensions

$$c(\Delta\mathbf{r}, \Delta t) = (4\pi\Delta t)^{-1/2} \exp\left[-\frac{\Delta r^2}{4D\Delta t}\right].$$

Particles that have a finite extent, such as spheres and cubes, are easily handled by considering their centroids as the diffusing point particles. Particles with orientational degrees of freedom are more difficult to handle and for now we focus on the sphere case.

Assume that the protective region $\mathcal{P}_i$ is disjoint from all other protective regions and the core $\mathcal{C}_i$ is restricted to remain within $\mathcal{P}_i$. For point particles, one can show that the probability $c(\Delta\mathbf{r}, \Delta t)$, *conditional* on the fact that the particle never leaves the interior of $\mathcal{P}_i$, is again a Green's function for the diffusion equation but with the additional boundary condition that $c$ vanish on the boundary of $\mathcal{P}_i$. A single-particle propagator consists of *sampling* from such a probability distribution $c_1$. For simple protective regions such as cubes or spheres relatively simple closed-form solutions for $c_1$ exist. The probability distribution $c_1$ is only valid under the assumption that a given particle $i$ remains inside $\mathcal{P}_i$. From $c_1$ (specifically, the flux $\boldsymbol{\nabla} c_1$ on the boundary of $\mathcal{P}_i$) one can also find the probability distribution that a particle first leaves $\mathcal{P}_i$ for the first time at a time $\tilde{t}$ and at position $\tilde{\mathbf{r}}$, i.e., the *first-passage* probability $J_1(\tilde{t}, \tilde{\mathbf{r}})$. This distribution can be used to sample a time at which particle $i$ is propagated to the surface of $\mathcal{P}_i$ and $\mathcal{P}_i$ is updated.

The basic idea of the AED DKMC algorithm is to protect the particles with disjoint protective regions and then use single-particle propagators to evolve the system. Typically the protective regions would have the same position and shape as the particle itself but be enlarged by a certain scaling factor $\mu_{\mathcal{P}} > 1$. At some point in time, however, two particles $i$ and $j$ will collide and thus cannot be protected with disjoint regions. Such nearly-colliding pairs are protected by a *pair protection region* $\mathcal{P}_{ij}$ (e.g., two intersecting spheres, each centered around one of the particles). A *pair propagator* $c_2(\Delta\mathbf{r}_i, \Delta\mathbf{r}_j, \Delta t)$ is used to either find the first-passage time, that is, the time when one of the particles leaves $\mathcal{P}_{ij}$, or to propagate the pair conditional on the fact that both particles remain inside $\mathcal{P}_{ij}$. Analytical solutions for $c_2 = c_2^{(D)} c_2^{(CM)}$ can be found by splitting the problem into independent diffusion problems for the center of (diffusional) mass $\mathbf{r}_{ij}^{(CM)} = (\Delta\mathbf{r}_i + \Delta\mathbf{r}_j)/2$ and for the difference $\mathbf{r}_{ij}^{(D)} = \Delta\mathbf{r}_i - \Delta\mathbf{r}_j$ walker (with some additional weighting factors for unequal particles). The condition for collision $d_{ij} = D_{ij}$ forms an additional absorbing boundary for the difference

walker, and a collision occurs whenever the first-passage propagator $J_2^{(D)}(\tilde{\mathbf{r}}_{ij}^{(D)}, \tilde{t}^{(D)})$ samples a point on that boundary. For repulsive particles [15] the boundary $d_{ij} = D_{ij}$ would be reflective (zero-flux) instead of absorbing.

If the particles are cubes closed-form solutions can easily be found for the pair propagators [13], however, in general, two-body propagators are considerably more complex (and thus costly) than single-body ones. One can in fact use approximate numerical propagators in which the difference walker takes small hops until it gets absorbed at one of the boundaries of its protection. This is similar to how one numerically predicts pair collisions between non-spherical particles in MD simulations using time-stepping for just a single pair of particles [4]. We have implemented hopping pair propagators for spheres. The only difference with the analytical propagators is that the hopping trajectory may need to be reversed later if a third particle forces a destruction of the pair protection before the scheduled pair event. We have used a state-saving mechanism in which the state of the random number generator at the beginning of the pair event is saved and later restored if the predicted event is cancelled. If the predicted even does in fact occur we avoid the cost of repeating the same hopping trajectory by saving the final state of the walk (i.e., the position of the difference walker) when predicting pair events.

### *1. AED Implementation of DKMC*

Geometric near-neighbor searches are an essential component of the DKMC algorithm, and the same methods (LLCs and NNLs) as in MD are used. Cell transfers are not explicitly predicted or processed in this algorithm, rather, whenever the position of a particle is updated the LLCs need to be updated accordingly. When NNLs are used, the collision of $\mathcal{C}_i$ with $\mathcal{N}_i$ may be sampled exactly, or, alternatively, the neighborhood $\mathcal{N}_i$ may be updated whenever $\mathcal{C}_i$ is very close to the inner wall of $\mathcal{N}_i$. We say that a particle $i$ is *protected against* particle $j$ or pair $jk$ if $\mathcal{P}_i$ is disjoint from $\mathcal{P}_j$ or $\mathcal{P}_{jk}$ (an unprotected particle has $\mathcal{P}_i \equiv \mathcal{C}_i$), similarly for pairs of particles. The goal of neighbor searches is to protect a particle $i$ against other particles and pairs with the largest possible $\mathcal{P}_i$. There is a balance between rebuilding protective and neighborhood regions too often and propagating the particles over smaller steps, and some experimentation is needed to optimize the algorithm and minimize the number of neighbor searches that need to be performed. Whenever a protection $\mathcal{P}_i$ is destroyed,

particle $i$ should be inserted back into the event queue with $t_e = t$, $p = \infty$, $\nu = 0$, so that it is protected again right away.

The main AS steps in Algorithm 1 for DKMC simulations are:

- Steps 2 and 3 may involve the processing of particle birth processes, where a particle of a given specie is introduced into the system to model external fluxes. These are typically assumed to occur as a Poisson process and therefore the time to the next birth is simply an exponentially distributed number, with the total birth rate given as the sum of the birth rates for each of the species. The birth process may be spatially homogeneous or the rate may depend on the cell in which the birth occurs. The newborn particles are inserted into the queue with $p = 0$, $\nu = -1$.

- Step 6 consists of sampling $\mathbf{r}_i$ from $c_1$ or $J_1$ and typically also rebuilding $\mathcal{P}_i$ as in step 8a. For $\nu > 0$, a particle decay reaction may be processed, i.e., particle $i$ may disappear to produce zero, one or two "product" particles, which are inserted into the queue with $p = 0$, $\nu = -1$.

- Step 7 consists of sampling positions $\mathbf{r}_i$ and $\mathbf{r}_j$ from the appropriate distribution:

  - If the event is the decay of $i$ or $j$, then $c_2^{(D)}$ and $c_2^{(CM)}$ are sampled, and then the decay reaction is processed.
  - If the event is the collision of $i$ and $j$, then $c_2^{(CM)}$ and $J_2^{(D)}$ are sampled, and the appropriate reaction (e.g., annihilation, coalescence, chemical reaction, etc.) is processed. This may destroy $i$ and/or $j$ and/or create new particles to be inserted into the queue (with $p = 0$, $\nu = -1$).
  - If the event is the dissolution of the pair $ij$, then either $c_2^{(CM)}$ and $J_2^{(D)}$, or $c_2^{(D)}$ and $J_2^{(CM)}$ are sampled, particle $j$ is inserted back into the queue with $p = 0$, $\nu = 0$, and $\mathcal{P}_i$ is updated as in step 8a (this may protect the particles $i$ and $j$ as a pair again).

- Step 8a is the primary type of event in step 8 and consists of updating $\mathcal{P}_i$. The processing of such "virtual" collisions with $\mathcal{P}_i$ consists of searching for the nearest protection region $\mathcal{P}_j$ or $\mathcal{P}_{jk}$ among the neighbors of particle $i$ (either using LLCs or NNLs). Particle $i$ is then protected against that nearest neighbor. If this makes $\mathcal{P}_i$

too small then particle $j$ or pair $jk$ is propagated to time $t$ and $\mathcal{P}_j$ or $\mathcal{P}_{jk}$ destroyed. Finally, $\mathcal{P}_i$, $\mathcal{P}_{ij}$ or $\mathcal{P}_{ik}$ is constructed again, depending on the exact local geometry.

- Step 9 consists of sampling event times from the appropriate distributions:
  - In step 9a $J_1$ is sampled.
  - In step 9b an exponentially distributed time is generated based on the decay rates for specie $s_i$.
  - In step 9c $J_2$ is sampled, as well as a decay time for each of the two particles, and the earliest of the three times is selected.

*2. Time-Driven Handling*

Under certain conditions the exact event-driven handling of diffusion may become inefficient or cumbersome. For example, since all protections are either of single particles or pairs very small protective regions need to be used in very dense clusters of particles where there may be many nearly colliding triplets of particles. The hops taken by the asynchronous algorithm will then become just as small as what a time-driven (approximate) algorithm would use, and it will therefore be more efficient to use time-stepping for those particles (and thus avoid the cost of event queue operations and also simplify overlap search). As another example, some particles may have complex internal structure and thus their diffusion or interactions with other particles may be difficult to treat analytically. For example, tightly-bound collections of particles may act as a single particle that has complex internal structure and dynamics (relaxation). Even for spherical particles it has proven that exact pair propagators are difficult to implement.

Just as for the SEDMD algorithm above, one can add a time-driven component to the asynchronous event-driven algorithm. Particles that are time-driven do not need to be protected against each other, instead, they may be unprotected or they can be protected only against event-driven particles. Particles whose protective regions overlap form a cluster and should be treated using a synchronous algorithm (this cluster may include all time-driven particles). In diffusion problems it is often the case that different species have widely differing diffusion coefficients and therefore very different time-steps will be appropriate for

different species. To solve this problem, one can use the $n$-fold (BKL) synchronous event-driven algorithm inside each cluster. In this algorithm, at each event (hop) only particles of a single specie hop by a small but non-negligible distance and the rest remain in place, and more mobile particles are hopped more frequently (with the correct relative frequency) than the less mobile ones.

Note that a cluster may freely take hops up to the time of the next event in the queue without stopping and modifying the event queue, since it is known that those hops will not be pre-empted by another event. In some situations this may improve efficiency by reducing the number of heap operations and also increasing the locality of the code by focusing multiple events on the same (cached) small group of particles.

## IV. DISCUSSION

In this section we focus on some of the difficulties in deploying AED algorithms in the simulation of realistic systems. The best-known difficulty is the parallelization of AED algorithms [6], which we do not discuss here due to space limitations. Instead, we will focus on the difficulties that make even serial simulations challenging. It is important to point out that for problems involving hard particles, that is, particles interacting with discontinuous potentials, event-driven approaches are the only exact algorithm. Time-driven algorithms always make an error due to the finite size of the time step $\Delta t$, and typically $\Delta t$ must be much smaller than the actual time step between events in order to guarantee that no events are missed.

The most involved aspect of implementing an AED algorithm for a particular problem is the need for analytic solutions for various one- and two-body propagators or event predictions. For MD, the difficulty is with predicting the time of collision of two moving hard particles. For hard spheres, this can be done analytically relatively easily (but numerical care must be taken [8]). When orientational degrees of freedom are involved, however, a time stepped ODE-like methodology is needed since analytical solutions are difficult to obtain [4]. This makes collision prediction much more difficult to implement in a numerically-stable way and also much more costly. In DKMC, there is a need to analytically construct the probability distributions $c_1$, $c_2$, $J_1$ and $J_2$, or at least to find a way to efficiently sample from them. These distributions are Green's functions for a time-dependent diffusion equation inside re-

gions such as spheres, cubes, spherical shells, intersection of two spheres, or intersection of a cone and a sphere. For time-independent problems such solutions can be constructed more easily, but for time-dependent problems even the simple diffusion equation poses difficulties (analytical solutions are typically infinite series of special functions in the Laplace domain). Different boundary conditions such as reactive surfaces require even more analytical solutions and tailor-made propagators. The handling of more complex equations of motion such as the full Langevin equation (which combines convection and diffusion) has not even been attempted yet.

This makes designing a more general-purpose AED program virtually impossible. This is to be contrasted to, for example, time-driven MD where different interaction potentials can used with the same time integrator. In general, time stepped approaches are the only known way to solve problems for which analytical solutions do not exist, including two-body problems in the case of MD or DKMC. The algorithm used in Ref. [4] to predict the time of collision for pairs of hard ellipsoids combines a time-driven approach with the event-driven one. It does this without trying to combine them in an intelligent way to avoid wasted computation (such as repeated trial updates of the position of a given particle as each of its neighbors is processed). We believe that such an intelligent combination will not only provide a more general AED algorithm, but also make the algorithm more robust numerically.

More generally, combining event-driven with time-driven algorithms is important for efficiency (at a certain sacrifice in accuracy). When the time step is large enough so that many events occur within one time step one can use the event-driven algorithm in-between the updates in the time-driven approach [8, 17, 20]. When the time step is small, however, events occur sparingly only in some of the time steps, and a different methodology is needed. For example, a new kind of *time step event* can be added that indicates propagation over a small time step (e.g., for the set of particles in pure DSMC cells). The essential advantage of event-driven algorithms is that they automatically adjust to the time scale at hand, that is, that they take the appropriate time step without any additional input. The real challenge is to use time stepping in an event-driven framework in which the time step is adjusted accordingly, while still processing events correctly. We have described such a framework for the SEDMD and DKMC algorithms.

Another unexplored or barely explored area is that of using controlled approximations in AED algorithms. Approximate event-driven algorithms have been used to handle a variety

of processes, however, these are often uncontrolled approximations. The approximate algorithms may reproduce the required (macroscopic) physical averages just as well as the exact algorithm would, however, controls are necessary to validate the simulations. Examples of approximations that may be useful include ignoring unlikely interactions between certain particles, approximate solutions instead of exact propagators (such as expansions around the mean behavior), etc.

Almost all of the AED particle algorithms to date have focused on single-particle or pair events. This is possible to do for hard particles because exact triple collisions are a zero-measure event. However, for more realistic models, or when approximations are made, events involving clusters of particles may need to be considered. For example, a cluster of particles may evolve as a strongly-coupled (e.g., chemically bonded) unit while interacting with other (e.g., freely diffusing) single particles or clusters. An additional assumption in most AED particle algorithms is that events affect only one or two particles, so that the event predictions of the majority of particles remain valid after processing an event. In some situations, however, there may be global degrees of freedom and associated events that affect all of the particles. For example, in MD there may be a macroscopic strain rate that affects all of the particles, since all of the event predictions are invalidated when the strain rate changes. And in principle the strain rate is coupled back to each of the particles, so that every particle collision also changes the strain rate (albeit by a small amount). In time-driven MD this is no problem since the evolution of the system is synchronous and the strain rate evolves together with the particles, however, in event-driven MD such coupling between all of the particles makes it impossible to schedule events efficiently.

Finally, multi-algorithm and/or multi-scale combinations including an AED component have not been explored to our knowledge. As an example, consider the simulation of nano-structures during epitaxial film growth [14]. At the smallest scales, time-driven (first-principles or classical) MD is needed in order to study the attachment, detachment, or hopping of individual particles or clusters. Once the rates for these processes are known, lattice-based KMC can be used to evolve the structure more quickly without simulating the detailed (vibrational) motion of each atom. At larger scales, the continuum-based DKMC algorithm we described can be used to propagate atoms over large distances in lower-density regions (across flat parts of the surface). Finally, a time-driven continuum diffusion can be used to model processes at macroscopic length scales. Such ambitious investigations are a

challenge for the future.

## V. ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48 (UCRL-JRNL-???).

---

[1] F. J. Alexander and A. L. Garcia. The Direct Simulation Monte Carlo Method. *Computers in Physics*, 11(6):588–593, 1997.

[2] A. Donev, B. J. Alder, and A. L. Garcia. An Event-Driven Hybrid Molecular Dynamics and Direct Simulation Monte Carlo Algorithm. In preparation, 2007.

[3] A. Donev, I. Cisse, D. Sachs, E. A. Variano, F. H. Stillinger, R. Connelly, S. Torquato, and P. M. Chaikin. Improving the Density of Jammed Disordered Packings using Ellipsoids. *Science*, 303:990–993, 2004.

[4] A. Donev, S. Torquato, and F. H. Stillinger. Neighbor List Collision-Driven Molecular Dynamics Simulation for Nonspherical Particles: I. Algorithmic Details II. Applications to Ellipses and Ellipsoids. *J. Comp. Phys.*, 202(2):737–764, 765–793, 2005.

[5] J. Elf, A. Doncic, and M. Ehrenberg. Mesoscopic reaction-diffusion in intracellular signaling. In S. M. Bezrukov, H. Frauenfelder, and F. Moss, editors, *Fluctuations and Noise in Biological, Biophysical, and Biomedical Systems*, pages 114–124, 2003.

[6] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, 2000.

[7] H. J. Herrmann and M. Müller. Simulations of granular materials on different scales. *Comp. Phys. Comm.*, 127:120–125, 2000.

[8] Y. A. Houndonougbo, B. B. Laird, and B. J. Leimkuhler. Molecular dynamics algorithms for mixed hard-core/continuous potentials. *Mol. Phys.*, 98:309–316, 1999.

[9] F. Jamalyaria, R. Rohlfs, and R. Schwartz. Queue-based method for efficient simulation of biological self-assembly systems. *J. Comput. Phys.*, 204(1):100–120, 2005.

[10] H. Karimabadi, J. Driscoll, Y. Omelchenko, and N. Omidi. A new asynchronous methodology for modeling of physical systems: breaking the curse of courant condition. *J. Comp. Phys.*, 205(2):755–775, 2005.

[11] B. Mirtich. Timewarp rigid body simulation. In *SIGGRAPH '00: Proceedings of the 27th*

*annual conference on Computer graphics and interactive techniques*, pages 193–200, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[12] M. A. Novotny. *A Tutorial on Advanced Dynamic Monte Carlo Methods for Systems with Discrete State Spaces*, volume IX of *Annual Reviews of Computational Physics*, pages 153–210. World Scientific, Singapore, 2001.

[13] T. Oppelstrup, V. V. Bulatov, G. H. Gilmer, M. H. Kalos, and B. Sadigh. First-passage monte carlo algorithm: Diffusion without all the hops. *Physical Review Letters*, 97(23):230602, 2006.

[14] J. S. Reese, S. Raimondeau, and D. G. Vlachos. Monte Carlo Algorithms for Complex Surface Reaction Mechanisms: Efficiency and Accuracy. *J. Comp. Phys.*, 173(1):302–321, 2001.

[15] A. Scala, T. Voigtmann, and C. D. Michele. Event-Driven Brownian Dynamics for Hard Spheres. *J. Chem. Phys*, 126(13):134109.

[16] K. E. Schmidt, P. Niyaz, A. Vaught, and M. A. Lee. Green's function Monte Carlo method with exact imaginary-time propagation. *Phys. Rev. E*, 71(1):016707, 2005.

[17] H. Sigurgeirsson, A. Stuart, and W.-L. Wan. Algorithms for Particle-Field Simulations with Collisions. *J. Comp. Phys.*, 172:766–807, 2001.

[18] S. W. Smith, C. K. Hall, and B. D. Freeman. Molecular Dynamics for Polymeric Fluids Using Discontinuous Potentials. *J. Comp. Phys.*, 134(1):16–30, 1997.

[19] J. S. van Zon and P. R. ten Wolde. Green's-function reaction dynamics: A particle-based approach for simulating biochemical networks in time and space. *The Journal of Chemical Physics*, 123(23):234910, 2005.

[20] R. S. Wedemann, V. C. Barbosa, and R. Donangelo. Defeasible time-stepping. *Parallel Comput.*, 25(4):461–489, 1999.